1 Pre-Check

This section is designed as a conceptual check for you to determine if you conceptually understand and have any misconceptions about this topic. Please answer true/false to the following questions, and include an explanation:

1.1 The t registers can be changed after calling a function and having that function return, while a registers cannot.

1.2 Let a0 point to the start of an array x. lw s0, 4(a0) will always load x[1] into s0.

1.3 Assuming no compiler or operating system protections, it is possible to have the code jump to data stored at 0(a0) and execute instructions from there.

1.4 Adding the character ’d’ to the address of an integer array would get you the 26th element of that array (assuming the array is large enough).

1.5 Calling jalr is a shorthanded expression for jal that jumps to the specified label and does not store a return address anywhere.

1.6 Calling j label does the exact same thing as calling jal label.
2 RISC-V with Arrays and Lists

Comment what each code block does. Each block runs in isolation. Assume that there is an array, int arr[6] = {3, 1, 4, 1, 5, 9}, which starts at memory address 0xBFFFFF00, and a linked list struct (as defined below), struct ll* lst, whose first element is located at address 0xABCD0000. Let s0 contain arr's address 0xBFFFFF00, and let s1 contain lst's address 0xABCD0000. You may assume integers and pointers are 4 bytes and that structs are tightly packed. Assume that lst's last node's next is a NULL pointer to memory address 0x00000000.

```c
struct ll {
    int val;
    struct ll* next;
}
```

2.1

```c
lw  t0, 0(s0)
lw  t1, 8(s0)
add t2, t0, t1
sw  t2, 4(s0)
```

2.2

```c
loop: beq  s1, x0, end
    lw  t0, 0(s1)
    addi t0, t0, 1
    sw  t0, 0(s1)
    lw  s1, 4(s1)
    jal  x0, loop
end:
```

2.3

```c
add  t0, x0, x0
loop: slti t1, t0, 6
    beq  t1, x0, end
    slli t2, t0, 2
    add  t3, s0, t2
    lw  t4, 0(t3)
    sub  t4, x0, t4
    sw  t4, 0(t3)
    addi t0, t0, 1
    jal  x0, loop
end:
```

3 RISC-V Calling Conventions

3.1 How do we pass arguments into functions?
3.2 How are values returned by functions?

3.3 What is sp and how should it be used in the context of RISC-V functions?

3.4 Which values need to saved by the caller, before jumping to a function using jal?

3.5 Which values need to be restored by the callee, before returning from a function?

3.6 In a bug-free program, which registers are guaranteed to be the same after a function call? Which registers aren’t guaranteed to be the same?
4 Writing RISC-V Functions

4.1 Write a function \texttt{sumSquare} in RISC-V that, when given an integer \( n \), returns the summation below. If \( n \) is not positive, then the function returns 0.

\[
n^2 + (n-1)^2 + (n-2)^2 + \ldots + 1^2
\]

For this problem, you are given a RISC-V function called \texttt{square} that takes in a single integer and returns its square.

First, let’s implement the meat of the function: the squaring and summing. We will be abiding by the caller/callee convention, so in what register can we expect the parameter \( n \)? What registers should hold \texttt{square}’s parameter and return value? In what register should we place the return value of \texttt{sumSquare}?

4.2 Since \texttt{sumSquare} is the callee, we need to ensure that it is not overriding any registers that the caller may use. Given your implementation above, write a prologue and epilogue to account for the registers you used.
5 More Translating between C and RISC-V

5.1 Translate between the RISC-V code to C. What is this RISC-V function computing? Assume no stack or memory-related issues, and assume no negative inputs.

<table>
<thead>
<tr>
<th>C</th>
<th>RISC-V</th>
</tr>
</thead>
<tbody>
<tr>
<td>// a0 -&gt; x, a1 -&gt; y,</td>
<td>Func: addi t0 x0 1</td>
</tr>
<tr>
<td>// t0 -&gt; result</td>
<td>Loop: beq a1 x0 Done</td>
</tr>
<tr>
<td></td>
<td>mul t0 t0 a0</td>
</tr>
<tr>
<td></td>
<td>addi a1 a1 -1</td>
</tr>
<tr>
<td></td>
<td>jal x0 Loop</td>
</tr>
<tr>
<td></td>
<td>Done: add a0 t0 x0</td>
</tr>
<tr>
<td></td>
<td>jr ra</td>
</tr>
</tbody>
</table>